Санкт-Петербургский государственный университет
Кафедра компьютерного моделирования и многопроцессорных систем

Бендриковский Анатолий Ярославович

Выпускная квалификационная работа бакалавра

Выделение объекта из видеопотока с помощью глубинного обучения

Направление 010400

Прикладная математика, фундаментальная информатика и основы программирования

Научный руководитель,
Ph.D.,
доцент
Корхов В. В.

Санкт-Петербург

2017

Содержание

Введение

На сегодняшний день человечество шагнуло далеко вперед – появляется все больше сфер, где работа человека заменяется искусственным интеллектом.

Причин на то много: монотонность труда, человеческий фактор, неспособность конкурировать по скорости реакции, а в некоторых случаях неприспособленность нашего организма, наших органов чувств для работы в нестандартных ситуациях.

Например, управление автомобилем в условиях недостаточной видимости (тумана, снегопада, сумерек) может привести к дорожно-транспортным происшествиям, автопилот же может использовать лидары и различные датчики, чтобы «видеть сквозь осадки», при этом также не может быть ослеплен ярким солнцем. Уже сейчас существуют автомобили, способные управлять автомобилем на шоссе, с хорошей разметкой и видимостью дороги. В скором будущем автопилот будет управлять автомобилем безопаснее и лучше, чем человек. К тому же, автопилот может связываться с облачными сервисами и использовать данные для прокладки маршрута, избегая заторов на дорогах, поиска парковочного места и многое другое.

Одной из важнейших задач для реализации автопилота, является анализ изображения, а точнее – онлайн обработка видеопотока с камеры.

На сегодняшний день актуальной задачей в области цифровой обработки изображений является обнаружение и классификация объектов, которая подразумевает собой заключение объектов в обрамляющие окна и присвоение класса. При достижении высокой скорости работы алгоритма можно приблизиться к обработке видеопотока, представленного в виде набора последовательных изображений.

Результаты достижений в этой сфере широко применяются в раз-

работке программного обеспечения для автопилотируемых технических средств, летательных аппаратов, всевозможных «ассистентов водителя», систем фиксации нарушения ПДД и во многих других направлениях.

Например, в дипломной работе Кулинкина А. Б. «Нейросетевое детектирование объектов в условиях ограниченного времени.» рассмотрена задача обнаружения предметов гардероба на изображении [1].

На мой взгляд, наиболее интересными и востребованными направлениями являются автопилот, «ассистент водителя» и анализ изображения с камер видеонаблюдения, которые требуют нахождения различных объектов с такими классами, как пешеход, автомобиль, мотоцикл, велосипед, светофор, дорожные и номерные знаки.

Постановка задачи

От алгоритма обнаружения требуется высокая точность классификации объекта и определения его местоположения на изображении, важна также и скорость обработки изображения. Алгоритмы на основе глубоких нейросетей сейчас достигают очень хороших показателей по данным критериям. Спектр подходов к решению этой задачи достаточно велик, что позволяет выбирать наиболее подходящие под конкретные нужды методы.

Целью ВКР является изучение и сравнение существующих методов обнаружения и классификации изображений, выбор наиболее подходящего нейросетевого алгоритма для работы применительно к задаче автопилота.

Для достижения обозначенной цели были сформулированы следующие задачи:

- Рассмотрение и сравнение алгоритмов, являющихся представителями различных подходов;
- Проведение анализа их точности и скорости работы;
- Выбор наиболее подходящего алгоритма для работы применительно к задаче автопилота.
  - Более тонкая настройка выбранного алгоритма для обнаружения объектов, которые играют важную роль в этой задаче: пешеход, автомобиль, мотоцикл, велосипед, красный и зеленый светофоры;
  - Обработка с помощью него видеофрагментов.

Обзор литературы

При написании данной работы были использованы различные научные статьи, публикации в различных научных изданиях.

Основными источниками, раскрывающими принципы работы алгоритмов, основанных на предположениях регионов (region proposals) являются статьи [2] и [3]. В этих работах описаны архитектура алгоритмов и детали их реализации, описаны основные улучшения по сравнению с алгоритмами-предшественниками, приведены сравнительные характеристики скорости и точности. Также описаны эксперименты на наборе данных PASCAL VOC 2007 test [4].

В презентации [4] и статье [5] приведена основная информация о соревновании «The PASCAL VOC Challenge», а в статье [6] о соревновании «Microsoft COCO: Common Objects in Context».

Работа [7] и [8] полностью раскрывают алгоритм YOLO, архитектуру нейронной сети, улучшения YOLOv2 по сравнению с YOLO и алгоритма для 9000 классов YOLO9000. А так же предоставляет сравнительный анализ алгоритмов Fast R-CNN, Faster R-CNN и YOLO на наборах данных [4] и [6].

Так же следует отметить полноту руководств [9] и работу [10], в которых раскрыты тонкости настройки алгоритма YOLO, а также официальный сайт [11].

Термины

Введем необходимые термины:

- Ground truth – правильные ответы, в нашем случае это обрамляющие окна, помеченные классами, которые были вручную нанесены на изображение.
- GPU (graphics processing unit) – графический процессор, эффективен для обработки и отображения компьютерной графики и выполнения большого количества несложных вычислений.
- CPU (central processing unit) –центральный процессор
- SVM(support vector machine) - метод опорных векторов, алгоритм обучения с учителем, который используется для задач классификации и регрессии.
- Frames per Second (FPS) – кадровая частота. Количество обрабатываемых за одну секунду кадров.

Глава 1. Оценки производительности и точности

Для оценки производительности и точности алгоритмов существует ряд оценок, вычислив их для какого-то конкретного набора данных, можно сравнить качество работы нескольких алгоритмов. В нашем случае, возникает проблема выбором нужной меры по причине того, что оценить нужно, как качество обнаружение объектов из определенного нами списка классов, так и правильная классификация этого объекта.

Для задачи обнаружения (detection) объектов обычно используется такая мера, как средняя точность (Average Precision(AP)), полученная из двух показателей: точности (precision) и полноты (recall).

Рассмотрим возможные события при обнаружении объекта какого-то одного конкретного класса:

- Истинно положительные события (true positives) – объект нужного класса был правильно заключен в обрамляющее окно – найден.

- Ложно положительные события (false positives) – в обрамляющее окно было заключено что-то другое, но не объект нужного класса.

Теперь перейдем к определению понятий полноты и точности. Они находятся отдельно для каждого класса, на которых обучен алгоритм.

1.1. Точность и полнота

Точность (precision) обозначает процент верных предсказаний для обрамляющего окна (true positives) среди всех результатов для этого класса.

$$Precision = \frac{True\ positives}{True\ positives + False\ positives}$$

Полнота (recall) обозначает процент найденных обрамляющих окон для данного класса, среди всех, представленных в ground truth этого изображения.

$$Recall = \frac{True\ positives}{Number\ of\ ground\ truth\ boxes}$$

1.2. Пересечение по объединению

Для того чтобы была возможность оценить правильность заключения объекта в обрамляющее окно используется метрика IoU. Типичное пороговое значение, указывающее на правильное обнаружение, составляет IoU $>$ 50%.

Рис. 1: Примеры IoU

Пересечение по объединению (Intersection over Union) – метрика, используемая для оценки точности работы алгоритмов, детектирующих объекты.

Значение IoU равно частному площади пересечения предсказанного обрамляющего окна и окна-ответа из ground truth на площадь объединения этих окон:

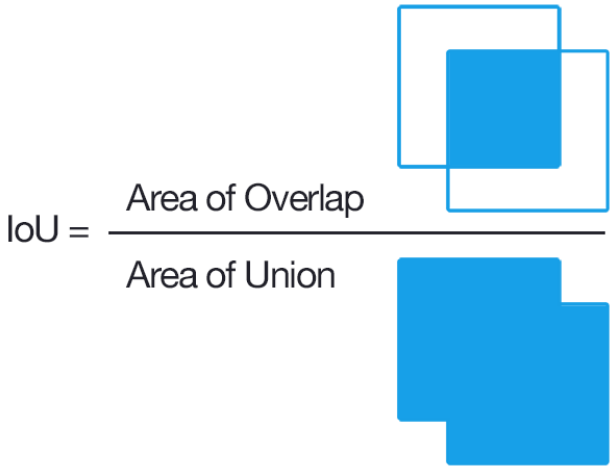

Рис. 2: IoU

1.3. Mean Average Precision

Поскольку мы производим оценку точности для алгоритма обнаружения и классификации, мы не можем ограничиться средней точность, поэтому в дальнейшем мы будем оперировать понятием Mean Average Precision.

Мера Mean Average Precision (mAP) – средняя величина AP

$$Mean\ Average\ Precision\ (mAP) = \frac{\sum_{q=1}^{Q} AP(q)}{Q},$$

где Q – количество запросов.

В нашем случае количество запросов соответствует количеству классов, на которые обучен алгоритм.

Глава 2. Подходы к решению задачи

Проблему обнаружение объектов на изображении можно рассматривать как задачу классификации или как задачу регрессии. В подходе на основе классификации изображение делится на участки, содержащие объект, отличный от фона. Каждый участок пропускается через классификатор, как самостоятельное изображение, чтобы определить класс объекта, изображенного на данном участке. В подходе использующем регрессию, все изображение пропускается через сверточную нейронную сеть, чтобы произвести обрамляющие окна для объектов на изображении. Рассмотрим алгоритмы использующие эти подходы.

2.1. Алгоритмы на основе предположений регионов

Алгоритмы Fast R-CNN, Faster R-CNN основаны на подходе region proposals, т.е. предположений регионов. Они состоят из двух частей, первая часть генерирует наборы участков, на которых вероятно изображен объект, отличный от фона. Вторая часть обрабатывает эти предположения и классифицирует объект на каждом участке.

Faster R-CNN [3] считается одним из самых точных нейросетевых алгоритмов обнаружения объектов на изображении, но для достижения высокой точности приходится жертвовать скоростью обработки, что не подходит для использования в онлайн-режиме.

2.2. Алгоритмы обрабатывающие изображение целиком

Представителем другого подхода к решению задачи обнаружения

объектов является алгоритм YOLO. В этом подходе нейронная сеть предсказывает обрамляющие окна (bounding boxes) и вероятности классов, применяясь к полному изображению. YOLO является одним из самых быстрых алгоритмов и подходит для работы в реальном времени, но имеет низкую точность на больших и маленьких объектах (средняя точность (mAP) — 63.4%).

Однако в улучшенной версии YOLOv2 [8] алгоритм достигает точности Faster R-CNN, не теряя в скорости.

Рассмотрим подробнее каждый из них и выясним какой подход лучше применительно к задаче автопилота.

Глава 3. Алгоритмы, основанные на предположении регионов

Рассмотрим понятие сверточной нейронной сети и нейросетевые алгоритмы на основе предположений регионов (region proposals), которые прошли путь от алгоритма R-CNN к алгоритму Faster R-CNN.

3.1. Сверточная нейронная сеть

CCNN (Convolutional Neural Network) – сверточная нейросеть. Отличие сверточной нейронной сети от обычной в том, что CNN содержит слои со сверточной архитектурой. В сверточном слое каждый узел соединен только с какой-то конкретной ограниченной областью предыдущего слоя - полем восприимчивости (receptive field). В случае обработки изображений, изображение представляет собой набор векторов размера три (R-red G-green B-blue) для каждого пикселя, поэтому устройство CNN таково, что каждый узел соединен с полем восприимчивости - квадратом размера $n \times n$ непосредственно на значениях пикселей. При этом это поля смещаются с некоторым фиксированным для всех узлов шагом и могут пересекаться.

На такую идею, ученых вдохновила работа Дэвида Хьюбела и Торстена Визела, связанная с изучением мозга обезьяны, в ходе которой было выяснено, что зрительная кора содержит клетки, восприимчивые только к поднабору клеток сетчатки и эти клетки пространственно близки.

3.2. R-CNN

Начнем с рассмотрения алгоритма R-CNN. Система обнаружения

объектов R-CNN работает в три этапа:

Первый этап генерирует порядка 2000 областей, которые с высокой долей вероятности содержат какой-либо объект, отличный от фона - region proposals. В R-CNN на этом этапе используется алгоритм Selective Search [12] или взаимозаменяемые.

Каждое из окон перемаштабируется и обрезается в изображение размера 227×227 пикселей и подается на вход CNN, которая в свою очередь пропускает его через пять сверточных слоев, два полносвязных слоя и выдает в виде вектора признаков размерности 4096.

Финальный этап представляет собой набор линейных SVM (support vector machine), которые причисляют каждый вектор какому-либо классу и отдельный регрессор обрамляющих окон.

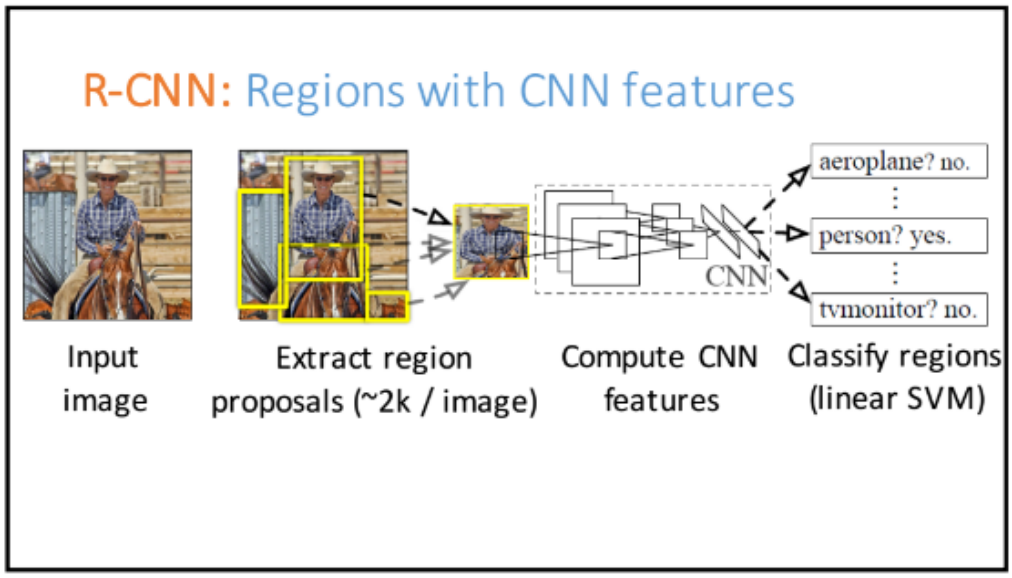

Рис. 3: Схема R-CNN

R-CNN имеет ряд недостатков, самый серьезный заключается в том, что требуется порядка двух-трех дней работы GPU, чтобы обучить алгоритм. При обработке требуется порядка 47 секунд на изображение, что тоже очень долго.

Таким образом, основное слабое место заключается в том, что каждое отдельное окно-кандидата требуется пропустить через CNN.

3.3. Fast R-CNN

Fast R-CNN предложен в 2015 году в статье [3] с улучшенным по сравнению с алгоритмом R-CNN временем обучения и временем обработки, а также с возросшей точностью обнаружения mAP с 62% до 66% на VOC 2012. Скорость обучения была увеличена в 9 раз, а скорость обработки в 213 раз, чем в оригинальной реализации R-CNN.

Нейросеть получает на вход изображение и набор окон-кандидатов. Несколько сверточных слоев и слоев max pooling производят карту признаков (feature maps).

Следующий слой RoI (region of interest) выделяет один вектор признаков из этой карты для каждого предположения объекта.

Этот вектор признаков в конечном итоге пропускается через серию полносвязных слоев и передается в два выходных слоя:

- слой softmax, производящий оценку вероятности по каждому классу;
- слой регрессора обрамляющих окон (bounding box regressor), который возвращает уточненные позиции обрамляющего окна.

Fast R-CNN показывает хорошие результаты для скорости обработки изображения, но он все еще опирается на медленные алгоритмы, генерирующие окна кандидаты. Selective Search [12] обычно требует порядка 1-2 секунд на изображение, тем самым ограничивая частоту обработки значением в 0.5 FPS, тем самым сводя на нет все дальнейшие улучшения скорости работы сверточной нейросети.

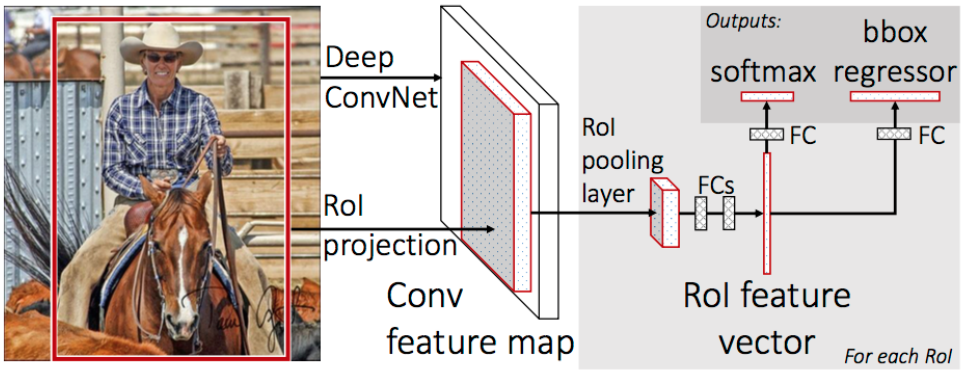

Рис. 4: Схема Fast R-CNN

	R-CNN	Fast R-CNN
Training Time:	84 hours	**9.5 hours**
(Speedup)	1x	**8.8x**
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**
mAP (VOC 2007)	66.0	**66.9**
Test time per image with Selective Search	50 seconds	**2 seconds**
(Speedup)	1x	**25x**

Рис. 5: Результаты Fast R-CNN в сравнении с R-CNN

3.4. Faster R-CNN

Алгоритм Faster R-CNN [3] является логическим продолжением алгоритма Fast R-CNN [2], в котором решены проблемы долгого и не очень точного поиска окон-кандидатов (Selective Search [12], MultiBox [13]). Тем самым, кроме значительного ускорения работы увеличилась средняя точность mAP с 66% до 70,4% на VOC 2012. Faster R-CNN использует для перового этапа собственную глубокую полносвязную нейросеть RPN (Region Proposal Networks), а для второго алгоритм Fast R-CNN.

Идея RPN основана на понимании того, что Fast R-CNN требует предположения по окнам-кандидатам только после первый сверточных слоев, а карты признаков, которые получаются после этих первых слоев, могут быть использованы для генерации окон-кандидатов. Таким образом большое количество вычислений может быть сокращено, если внести этап поиска регионов в нейросеть.

RPN строится путем добавления двух новых сверточных слоев поверх общих сверточных уровней Fast R-CNN. Так как большая часть вычислений в RPN используется совместно с сетью обнаружения, только стоимость двух дополнительных уровней влияет на скорость всего алгоритма Faster R-CNN.

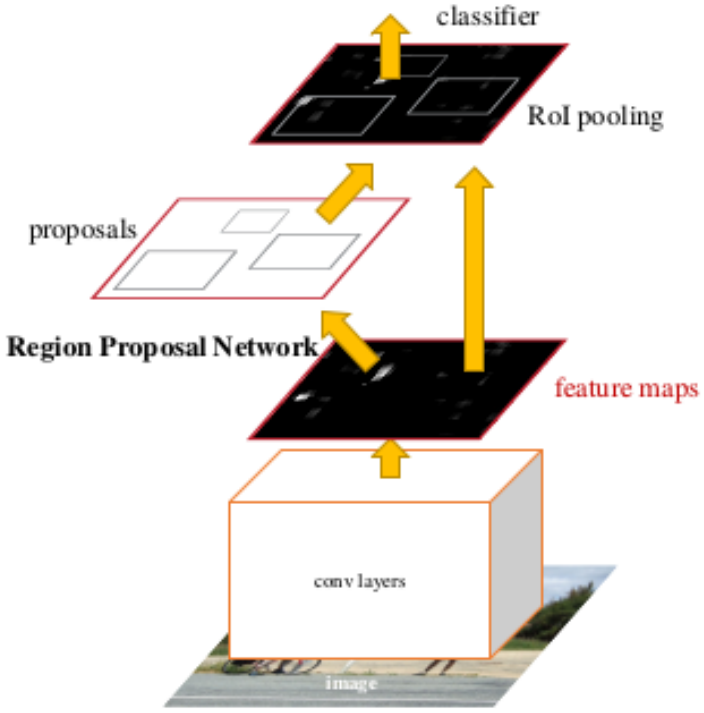

Рис. 6: Схема Faster R-CNN

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	**0.2 seconds**
(Speedup)	1x	25x	**250x**
mAP (VOC 2007)	66.0	**66.9**	**66.9**

Рис. 7: Сравнительные характеристики

Глава 4. YOLO

Рассмотрим алгоритм YOLO [7], который подходит для работы в реальном времени, но имеет низкую точность на больших и маленьких объектах, средняя точность (mAP) — 63,4%.

Однако в модификации YOLOv2 [8] точность достигает точности Faster R-CNN. Кроме того, существует Tiny YOLO версия, которая имеет меньшее количество слоев в нейросети, что позволяет использовать алгоритм на слабых устройствах с маленьким объемом памяти.

Так как в этом подходе нейронная сеть предсказывает обрамляющие окна (bounding boxes) и вероятности классов, применяясь к полному изображению – нейросеть «видит всю картину целиком». Это понимание контекста для объектов дает лучшие результаты при обработке картин, чем у других алгоритмов.

4.1. Описание алгоритма

На выходе нейросеть дает тензор размера $7 \times 7 \times 30$, который содержит всю информацию, необходимую для нанесения на изображение обрамляющих окон.

1. На изображение наносится сетка 7×7. Если центр объекта попадает в ячейку сетки, эта ячейка сетки отвечает за обнаружение этого объекта.

2. Каждой клетке этой сетки соответствует вектор размера $(4 + 1) \times B + C$, где:

   а) каждое обрамляющее окно (bbox — bounding box) характеризуется пятью показателями:

   - (x, y) — координаты центра соответствующей ячейки сетки;

- w — ширина обрамляющего окна относительно всего изображения;
- h — высота обрамляющего окна относительно всего изображения.
- некоторая вероятность того, что окно правильно нашло объект:

$$Pr(object) * IoU^{truth}_{predicted}$$

Здесь ground-truth bboxes — размеченные вручную обрамляющие окна тестирующего набора данных, predicted bounding boxes — предсказанные моделью окна;

b) B — число обрамляющих окон для каждой ячейки сетки, $B = 2$;

c) C — число классов, которые могут быть предсказаны. В каждом из этих элементов вектора хранятся вероятности того, что внутри ячейки сетки лежит объект конкретного класса, без привязки к конкретному обрамляющему окну.

3. Таким образом, каждая ячейка сетки предсказывает B ограничивающих окон и доверительные оценки (confidence scores) для них. Теперь перемножим эти оценки каждого окна с вероятностями классов соответствующей ячейки сетки. Получим набор векторов размера C в количестве $2 \times 7 \times 7$.

4. Для финального нанесения на изображение обрамляющих окон с полученным набором производятся следующие действия:

- рассматриваются значения набора векторов по классам;
- обнуляются не удовлетворяющие пороговому значению значения;
- сортируются по убыванию;

- применяется алгоритм Non-maximal suppressinon, который оставит наиболее точное из пересекающихся окон;
- на изображение наносятся только те окна, в векторе которых есть ненулевые значения, при этом окну присваивается класс с наибольшим значением.

4.2. Описание архитектуры нейросети

Изображение размера 448×448 пропускается через часть модифицированной модели классификации изображений GoogLeNet, которая в конечном счете имеет 24 сверточных слоя и 2 полносвязных слоя. Вместо «Inception module» GoogLeNet, используются слои 1×1 с последующим сверточными слоями 3×3.

В случае несоответствия размера изображения, оно будет растянуто под соответствующее разрешение.

Модель Tiny YOLO имеет 9 сверточных слоя и 3 полносвязных слоя. За счет меньшего количества параметров нейросети сокращается количество используемой памяти GPU, что дает возможность использовать алгоритм на встраиваемых системах, например на NVIDIA Jetson TX1. При этом точность mAP на VOC 2007 для YOLO составляет 63.4%, а для Tiny YOLO – 52.7%.

4.3. YOLOv2

В модификации YOLOv2 добавлены ряд улучшений, которые дают значительную прибавку в скорости и точности алгоритма, при этом точность достигла точности Faster R-CNN. Наиболее существенными улучше-

ниями являются:

- Нормализация партии (Batch Normalization) [14] дает прибавку точности в 2%. При обучении изображения подаются в нейросеть пачками, соответственно значения весов обновляются после обработки пачки. Нормализация партии приводит к существенному улучшению сходимости, устраняя необходимость в других формах регуляризации. Добавив нормализацию партии на всех сверточных уровнях в YOLO, получено более 2% улучшения в mAP. Нормализация партии также помогает упорядочить модель;

- High Resolution Classifier. Осуществлена точная настройка результирующей сети для обнаружения. Использован классификатор, обученный на наборе данных ImageNet. Эта сеть классификации с высоким разрешением дает прирост почти на 4% mAP;

- Dimension Clusters + Direct location prediction. Использование размерных кластеров наряду с прямым прогнозированием расположения центра обрамляющего окна улучшает YOLO почти на 5%.

Таким образом точность YOLO с 63,4% поднялась до 76,8 %.

В таблице представлены показатели алгоритмов, обученных на наборах данных VOC 2007 и VOC 2012, тестируемых на наборе test VOC 2007.

Алгоритмы	**Test**	**mAP**	**FPS**
Faster R-CNN VGG-16	test VOC2007	73,2	7
YOLOv2 416×416	test VOC2007	76,8	59
YOLO	test VOC2007	63,4	45

Глава 5. Модификация YOLO для задачи автопилота

Существует готовая реализация алгоритма YOLO, основанная на Darknet [15]. Darknet – это нейронная сеть с открытым исходным кодом, написанная на C и CUDA [16]. Она быстрая, простая в установке и поддерживает CPU и GPU вычисления.

Darknet предоставляет набор предобученных моделей для классификации ImageNet [17].

Для запуска алгоритма YOLO и Tiny YOLO можно воспользоваться обученными для 20-ти классов на различных наборах данных моделями и использовать их веса yolo.weights и tiny-yolo-voc.weights соответственно. Так как модели имеют различные конфигурации и набор слоев, они не могут быть заменены друг на друга.

Наша задача – изменить количество классов для обнаружения, и при этом внести новые классы, которых ранее не было в YOLO. Для этого мы должны получить свой файл с весами, так как yolo.weights обучен для конфигурации YOLO с 20-тью классами, он не подойдет для этой цели. Чтобы не обучать нейросеть с нуля, воспользуемся предобученными весами сверточных слоев darknet19_448.conv.23 [18].

Теперь изменим конфигурационный файл для нейросети. Изменим файл voc-yolo-2.0.cfg, который содержит в себе архитектуру улучшенной версии – YOLOv2. Назовем его obg-yolo.cfg.

В этом файле нужно изменить несколько показателей: classes=7, filters со 125 на 60. Это число соответствует выражению filters = (classes + 5)*5, где classes количество классов в новой модели.

Перечислим названия новых классов в файле obj.names:

- bicycle
- bus
- car
- motorbike
- person
- green traffic light
- red traffic light

Для обучения необходимо создать набор изображений, для которых существует соответствующий набор разметки. В данной реализации для изображения с названием, например car.jpg, должен существовать текстовый файл с таким же названием, но с расширением .txt, т.е. car.txt.

Этот файл должен быть заполнен в соответствии с форматом набора данных VOC 2007, т.е содержать в себе набор строк вида: «класс» «x» «y» «ширина» «высота».

- «класс» - натуральные числа от нуля до (classes-1).
- «x» «y» «ширина» «высота» - дробные числа, соответствующие ширине и высоте изображения от 0.0 до 1.0 - т.е. соответствуют масштабу всего изображения.
- «x» = «x координата» / «ширина изображения»
- «высота» = «высота объекта» / «высота изображения»
- «x» «y» – центр квадрата

Для того чтобы запустить обучение исходного алгоритма с 20-тью классами на наборе данных VOC2007 и VOC2012 нужно выполнить скрипт scripts/voc_label.py, чтобы сформировать изображения .jpg и .txt

Изменим этот скрипт (см. Приложение 1), чтобы сделать выборку необходимых нам классов car, bus, bicycle, motorbike, person из набора данных VOC 2007. Список будет храниться в файле train.txt в виде набора строк:

```
/home/darknet/scripts/VOCdevkit/VOC2007/JPEGImages/000012.jpg
```

Запустим обучение выполнив команду:

```
./darknet detector train data/obj.data cfg/yolo-obj.cfg
darknet19\_448.conv.23
```

В процессе обучения будут сохраняться промежуточные веса - каждые сто итераций до 1000, далее каждые 1000 итераций. Обучение на VOC 2007 длилось порядка 10ти часов в общей сложности. Это порядка 12000 итераций. Далее получившийся файл с весами можно протестировать на изображении:

```
./darknet detector test data/obj.data cfg/yolo-obj.cfg
backup/yolo-obj.backup data/dog.jpg
```

Видно, что объекты, присутствующие в нашем списке классов, обнаружены достаточно хорошо.

Но исходя из одного изображения невозможно сделать выводы о качестве и скорости работы алгоритма.

Выполним команду:

```
./darknet detector valid data/obj.data cfg/yolo-obj.cfg
backup/yolo-obj.backup
```

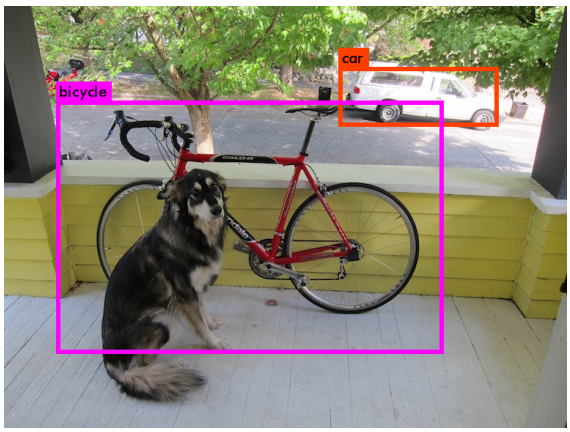

Рис. 8: Пример работы

Тестовая выборка сожержит изображения, которые не участвовали в этапе обучения. Если производительность измеряется данными, которые модель уже видела во время обучения, результатам нельзя доверять из-за общей проблемы машинного обучения, известной как переобучение. Переобучение - это когда модель изучает то, что было специфично для данных, показанных ранее, но которые не обобщаются на другие данные.

Обработка 2736 изображений заняла 63 секунды, что примерно равно 45.6 FPS.

Выполним команду для подсчета показателей точности:

```
./darknet detector recall data/obj.data cfg/yolo-obj.cfg
backup/yolo-obj.backup
```

```
2719   5461   6577       RPs/Img: 44.87   IOU: 67.55%     Recall:83.03%
2720   5462   6578       RPs/Img: 44.87   IOU: 67.56%     Recall:83.03%
2721   5463   6579       RPs/Img: 44.86   IOU: 67.56%     Recall:83.04%
2722   5463   6580       RPs/Img: 44.85   IOU: 67.56%     Recall:83.02%
2723   5464   6581       RPs/Img: 44.86   IOU: 67.56%     Recall:83.03%
2724   5465   6582       RPs/Img: 44.85   IOU: 67.56%     Recall:83.03%
2725   5470   6591       RPs/Img: 44.86   IOU: 67.54%     Recall:82.99%
2726   5471   6592       RPs/Img: 44.85   IOU: 67.55%     Recall:82.99%
2727   5473   6594       RPs/Img: 44.84   IOU: 67.55%     Recall:83.00%
2728   5475   6596       RPs/Img: 44.85   IOU: 67.55%     Recall:83.00%
2729   5476   6597       RPs/Img: 44.84   IOU: 67.55%     Recall:83.01%
2730   5477   6598       RPs/Img: 44.83   IOU: 67.55%     Recall:83.01%
2731   5479   6600       RPs/Img: 44.82   IOU: 67.55%     Recall:83.02%
2732   5482   6603       RPs/Img: 44.82   IOU: 67.55%     Recall:83.02%
2733   5483   6604       RPs/Img: 44.82   IOU: 67.56%     Recall:83.03%
```

Рис. 9: Показатели точности

Показатели полноты (recall) и пересечения по объединению (IoU) име-

ют достаточно высокие показатели, что говорит о хорошей точности алгоритма.

Дообучим полученные веса для новых классов красных и зеленых светофоров. Для этого воспользуемся набором данных [19]. Информация об объектах хранится в виде:

- frame
- xmin
- ymin
- xmax
- ymax
- occluded
- label
- attributes (Только для светофоров)

Приведем эти данные к формату VOC, оставив класс автомобилей и светофоров. Для запустим скрипт (см. Приложение 2). Добавим содержимое полученного файла в файл train.txt для предыдущего набора данных. После этого запустим алгоритм обучаться, предварительно перемешав все строки файла train.txt для того, чтобы не было переобучения на какой-то набор данных.

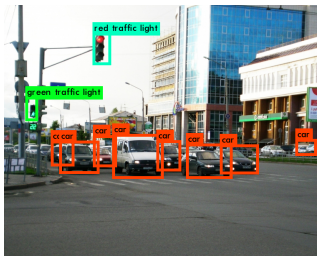

Рис. 10: Пример работы (см. Приложение 3)

Таким образом, алгоритм YOLO способен обрабатывать изображения с высокой скоростью и точностью. То есть, представив видеофрагмент в виде последовательного набора кадров, он будет обработан. Для этих целей используется библиотека OpenCV, произведем обработку видео с камеры видеорегистратора с помощью полученной нами модели.

Для этого выполним команду:

```
./darknet detector demo data/obj.data cfg/yolo-obj.cfg
backup/yolo-obj.backup video.mp4
```

Если не передавать во входные данные видеофрагмент, будет обработан видеопоток веб-камеры.

Заключение

В ходе работы были изучены алгоритмы обнаружения и классификации обхектов на изображении. Алгоритм YOLO обучен для новых классов, полезных для задачи автопилота. Алгоритм показывает хорошие результаты работы онлайн, оставаясь точным, что дает возможность обрабатывать видеофрагменты. Выполнена обработка видеофрагмента и видеопотока с веб-камеры. Больше примеров работы обученного на наших классах алгоритма YOLO можно увидеть в Приложении 3. Промежуточные результаты работы оформлены в виде научной статьи и будут опубликованы в одном из периодических изданий, входящих в Российский индекс научного цитирования.

Приложения

Приложение 1. Скрипт конвертации обучающих данных VOC 2007

```
import xml.etree.ElementTree as ET
import pickle
import os
from os import listdir, getcwd
from os.path import join

sets = [('2007', 'train'), ('2007', 'val'), ('2007', 'test')]

classes = ["bicycle", "bus", "car", "motorbike", "person", "traffic light", "sign"]

def convert(size, box):
    dw = 1. / (size[0])
    dh = 1. / (size[1])
    x = (box[0] + box[1]) / 2.0 - 1
    y = (box[2] + box[3]) / 2.0 - 1
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x * dw
    w = w * dw
    y = y * dh
    h = h * dh
    return (x, y, w, h)


def convert_annotation(year, image_id):
    in_file = open('VOCdevkit/VOC%s/Annotations/%s.xml' % (year, image_id))
    out_file = open('VOCdevkit/VOC%s/labels/%s.txt' % (year, image_id), 'w')
    tree = ET.parse(in_file)
    root = tree.getroot()
    size = root.find('size')
    w = int(size.find('width').text)
    h = int(size.find('height').text)
    count = 0
    for obj in root.iter('object'):
        difficult = obj.find('difficult').text
        cls = obj.find('name').text
        if cls not in classes or int(difficult) == 1:
            continue
        count += 1
        cls_id = classes.index(cls)
        xmlbox = obj.find('bndbox')
        b = (float(xmlbox.find('xmin').text),
         float(xmlbox.find('xmax').text), float(xmlbox.find('ymin').text),
             float(xmlbox.find('ymax').text))
        bb = convert((w, h), b)
        out_file.write(str(cls_id) + " " + " ".join([str(a) for a in bb]) + '\n')
    return count


wd = getcwd()

for year, image_set in sets:
    if not os.path.exists('VOCdevkit/VOC%s/labels/' % (year)):
        os.makedirs('VOCdevkit/VOC%s/labels/' % (year))
```

```
    image_ids = open('VOCdevkit/VOC%s/ImageSets/Main/%s.txt' % (year, image_set)).read().strip().split()
    list_file = open('%s_%s.txt' % (year, image_set), 'w')
    for image_id in image_ids:
        if(convert_annotation(year, image_id)>0):
            list_file.write('%s/VOCdevkit/VOC%s/JPEGImages/%s.jpg\n' % (wd, year, image_id))
        else:
            os.remove('VOCdevkit/VOC%s/labels/%s.txt'% (year, image_id))
    list_file.close()

os.system("cat 2007_train.txt 2007_val.txt > train.txt")
os.system("cat 2007_train.txt 2007_val.txt 2007_test.txt > train.all.txt")
```

Приложение 2. Скрипт конвертации обучающих данных Udacity

```
import xml.etree.ElementTree as ET
import pickle
import os
from os import listdir, getcwd
from os.path import join
import csv
from PIL import Image
from random import shuffle

classes = ["bicycle", "bus", "car", "motorbike", "person",
"green traffic light", "red traffic light", "sign"]

classes_in_udacity = ["car","pedestrian","trafficLight"]
def convert(size, box):
    dw = 1. / (size[0])
    dh = 1. / (size[1])
    x = (box[0] + box[1]) / 2.0 - 1
    y = (box[2] + box[3]) / 2.0 - 1
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x * dw
    w = w * dw
    y = y * dh
    h = h * dh
    return (x, y, w, h)

def new_data():

    global out_file
    last_name=" "
    wd = getcwd()
    if not os.path.exists('udacity/labels/'):
             os.makedirs('udacity/labels/' )
    list = []
    with open('udacity/JPEGImages/labels.csv','r') as f:
        csv_data = csv.reader(f, delimiter=' ', skipinitialspace=True)
        for data in csv_data:
            cls = data[6]
            image_name = data[0]
            if(cls in classes_in_udacity):
                if(last_name != image_name):
                    out_file = open('udacity/labels/%s.txt' % (image_name[0:len(image_name) - 4]), 'w')
                    list.append('%s/udacity/JPEGImages/%s\n' % (wd, image_name))
                else:
                    out_file = open('udacity/labels/%s.txt' % (image_name[0:len(image_name)-4]), 'a')

                b = (float(data[1]),float(data[3]),float(data[2]),float(data[4]))
```

```
                if(cls == 'trafficLight'):
                    if(len(data)>7):
                        color_of_light = data[7]
                        if(color_of_light == 'RedLeft' or color_of_light == 'Red'):
                            cls_id = classes.index("red traffic light")
                        else:
                            cls_id = classes.index("green traffic light")
                else:
                    if (cls == 'pedestrian'):
                        cls_id = classes.index('person')
                    else:
                        cls_id = classes.index(cls)
                bb = convert(Image.open("udacity/JPEGImages/%s" % (image_name)).size, b)
                out_file.write(str(cls_id) + " " + " ".join([str(a) for a in bb]) + '\n')
                last_name = image_name
    shuffle(list)
    train_file = open('udacity/train.txt', 'w')
    for i in range((int)(len(list)*0.8)):
        train_file.write(list[i])
    test_file = open('udacity/test.txt', 'w')
    for i in range((int)(len(list)*0.8),len(list)):
        test_file.write(list[i])
new_data()
```

Приложение 3. Примеры работы YOLO

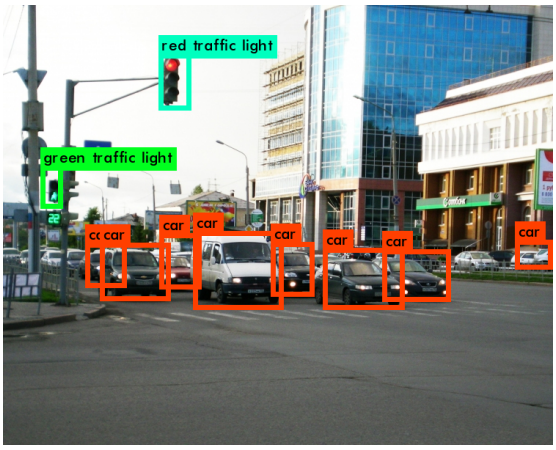

Рис. 11:

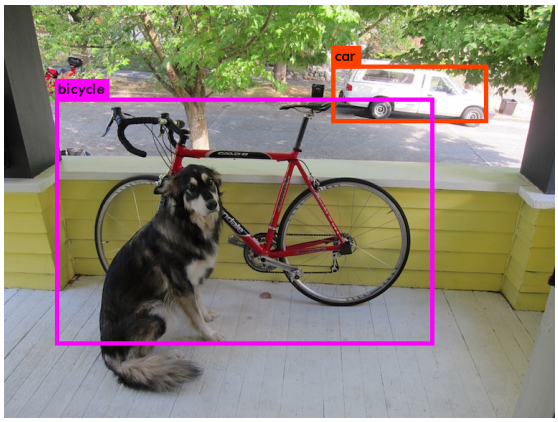

Рис. 12:

Рис. 13:

Рис. 14:

Рис. 15:

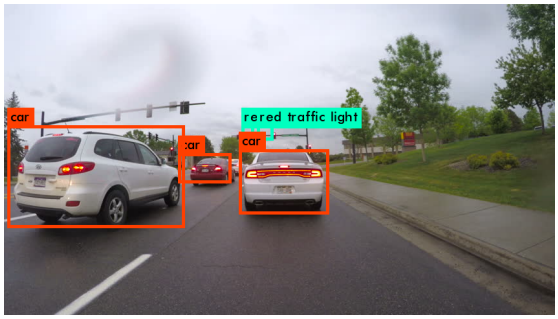

Рис. 16:

Список литературы

1. Кулинкин А. Б., Смирнов Е. А. Нейросетевое детектирование объектов в условиях ограниченного времени // Процессы управления и устойчивость. 2016. Т. 3.1. С. 419–424.

2. Girshick R. Fast R-CNN. https://arxiv.org/pdf/1504.08083

3. Shaoqing Ren S., He K., Girshick R., Sun J. Faster R-CNN. https://arxiv.org/pdf/1506.01497

4. The PASCAL VOC Challenge. http://www.frontiersincomputervision.com/slides/FCV_Dataset_Zisserman_2.pdf

5. The PASCAL VOC Challenge. http://arkitus.com/files/ijcv-14-everingham-pascal.pdf

6. Microsoft COCO: Common Objects in Context . http://arxiv.org/abs/1405.0312

7. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. https://arxiv.org/pdf/1506.02640

8. Redmon J., Farhadi A. YOLO9000: Better, Faster, Stronger. https://arxiv.org/pdf/1612.08242

9. Yolo Convolutional Neural Networks. https://github.com/AlexeyAB/darknet

10. Using 3D Graphics to Train Object Detection Systems. https://brage.bibsys.no/xmlui/bitstream/handle/11250/2418432/14189_FULLTEXT.pdf

11. YOLO: Real-Time Object Detection. https://pjreddie.com/darknet/yolo/

12. Uijlings J., K. van de Sande, Gevers T., Smeulders A. Selective Search for Object Recognition. https://ivi.fnwi. uva.nl/isis/publications/2013/ UijlingsIJCV2013/UijlingsIJCV2013.pdf

13. Szegedy C., Reed S., Erhan D., Anguelov D., Ioffe S. Scalable, High-Quality Object Detection. https://arxiv.org/pdf/1412.1441

14. Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/pdf/1502.03167

15. Darknet. https://pjreddie.com/darknet/

16. Compiling With CUDA. https://pjreddie.com/darknet/install/#cuda

17. ImageNet. http://www.image-net.org

18. Darknet weights. https://pjreddie.com/darknet/imagenet/#darknet19_448

19. Annotated Driving Dataset https://github.com/udacity/self-driving-car/tree/master/annotations